PXI9100 数据采集卡

WIN2000/XP 驱动程序使用说明书

阿尔泰科技发展有限公司
产品研发部修订

请您务必阅读《使用纲要》，他会使您事半功倍!

目　录

第一章 版权信息与命名约定

第一节、版权信息

第二节、命名约定

一、为简化文字内容，突出重点，本文中提到的函数名通常为基本功能名部分，其前缀设备名如 PXIxxxx_ 则被省略。如 PXI9100_CreateDevice 则写为 CreateDevice。

二、函数名及参数中各种关键字缩写

缩写	全称	汉语意思	缩写	全称	汉语意思
Dev	Device	设备	DI	Digital Input	数字量输入
Pro	Program	程序	DO	Digital Output	数字量输出
Int	Interrupt	中断	CNT	Counter	计数器
Dma	Direct Memory Access	直接内存存取	DA	Digital convert to Analog	数模转换
AD	Analog convert to Digital	模数转换	DI	Differential	(双端或差分) 注：在常量选项中
Npt	Not Empty	非空	SE	Single end	单端
Para	Parameter	参数	DIR	Direction	方向
SRC	Source	源	ATR	Analog Trigger	模拟量触发
TRIG	Trigger	触发	DTR	Digital Trigger	数字量触发
CLK	Clock	时钟	Cur	Current	当前的
GND	Ground	地	OPT	Operate	操作
Lgc	Logical	逻辑的	ID	Identifier	标识
Phys	Physical	物理的			

第二章 使用纲要

第一节、使用上层用户函数，高效、简单

如果您只关心通道及频率等基本参数，而不必了解复杂的硬件知识和控制细节，那么我们强烈建议您使用上层用户函数，它们就是几个简单的形如Win32 API的函数，具有相当的灵活性、可靠性和高效性。诸如InitDeviceProDA、WriteDeviceProDA等。而底层用户函数如WriteRegisterULong、ReadRegisterULong、WritePortByte、ReadPortByte……则是满足了解硬件知识和控制细节、且又需要特殊复杂控制的用户。但不管怎样，我们强烈建议您使用上层函数（在这些函数中，您见不到任何设备地址、寄存器端口、中断号等物理信息，其复杂的控制细节完全封装在上层用户函数中。）对于上层用户函数的使用，您基本上可以不必参考硬件说明书，除非您需要知道板上D型插座等管脚分配情况。因为上层函数的命名、参数的命名极其规范。

第二节、如何管理 PXI 设备

由于我们的驱动程序采用面向对象编程，所以要使用设备的一切功能，则必须首先用CreateDevice函数创建一个设备对象句柄hDevice，有了这个句柄，您就拥有了对该设备的绝对控制权。然后将此句柄作为参数传递给其他函数，如InitDeviceProDA可以使用hDevice句柄以程序查询方式初始化设备的DA部件，WriteDeviceProDA函数可以用hDevice句柄实现对DA数据的采样读取。最后可以通过ReleaseDevice将hDevice释放掉。

第三节、如何实现 DA 波形数据输出

当您有了hDevice设备对象句柄后，便可用InitDeviceProDA函数初始化DA部件，关于频率等参数的设置是

由这个函数的pDAPara参数结构体决定的。您只需要对这个pDAPara参数结构体的各个成员简单赋值即可实现所有硬件参数和设备状态的初始化。然后调用WriteDeviceProDA将准备好的DA数据写入板载RAM中，接着用StartDeviceProDA即可启动DA部件，开始DA输出。当您需要暂停设备时，执行StopDeviceProDA，当您需要关闭DA设备时，ReleaseDeviceDA便可帮您实现（但设备对象hDevice依然存在）。

第四节、哪些函数对您不是必须的

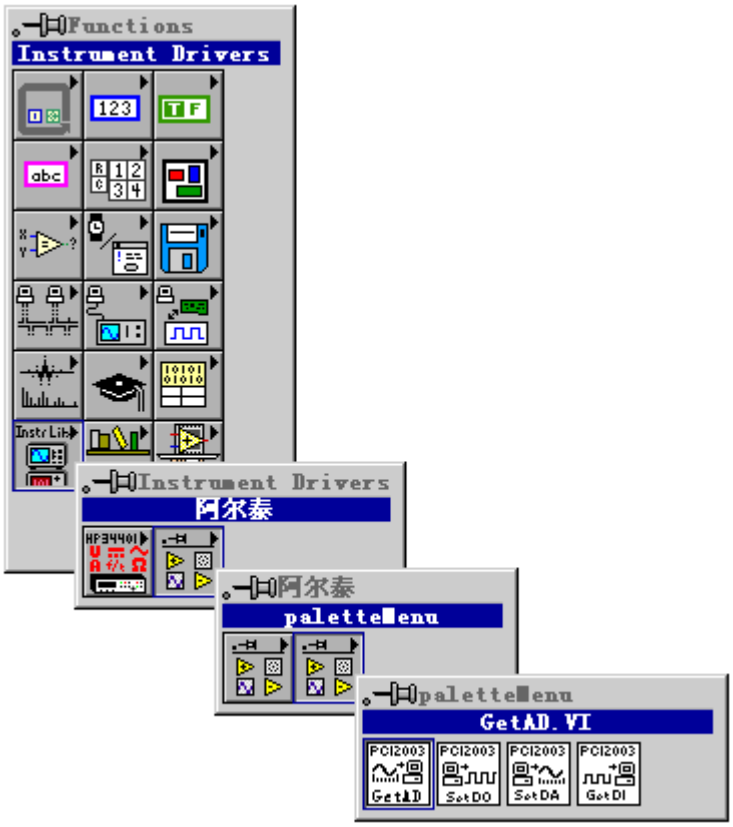
公共函数如CreateFileObject， WriteFile，ReadFile等一般来说都是辅助性函数，除非您要使用存盘功能。如果您使用上层用户函数访问设备，那么GetDeviceAddr， WriteRegisterByte， WriteRegisterWord，WriteRegisterULong， ReadRegisterByte， ReadRegisterWord， ReadRegisterULong等函数您可完全不必理会，除非您是作为底层用户管理设备。而WritePortByte， WritePortWord， WritePortULong， ReadPortByte，ReadPortWord， ReadPortULong则对PXI用户来讲，可以说完全是辅助性，它们只是对我公司驱动程序的一种功能补充，对用户额外提供的，它们可以帮助您在NT、Win2000 等操作系统中实现对您原有传统设备如ISA卡、串口卡、并口卡的访问，而没有这些函数，您可能在基于Windows NT架构的操作系统中无法继续使用您原有的老设备。

第三章 PXI 即插即用设备操作函数接口介绍

由于我公司的设备应用于各种不同的领域，有些用户可能根本不关心硬件设备的控制细节，只关心通道、采样频率等，然后就能通过一两个简易的采集函数便能轻松得到所需要的数据。这方面的用户我们称之为上层用户。那么还有一部分用户不仅对硬件控制熟悉，而且由于应用对象的特殊要求，则要直接控制设备的每一个端口，这是一种复杂的工作，但又是必须的工作，我们则把这一群用户称之为底层用户。因此总的看来，上层用户要求简单、快捷，他们最希望在软件操作上所要面对的全是他们最关心的问题。而关于设备的物理地址、端口分配及功能定义等复杂的硬件信息则与上层用户无任何关系。那么对于底层用户则不然。他们不仅要关心设备的物理地址，还要关心虚拟地址、端口寄存器的功能分配，甚至每个端口的Bit位都要了如指掌，看起来这是一项相当复杂、繁琐的工作。但是这些底层用户一旦使用我们提供的技术支持，则不仅可以让您不必熟悉PXI总线复杂的控制协议，同是还可以省掉您许多繁琐的工作，比如您不用去了解PXI的资源配置空间、PNP即插即用管理，而只须用GetDeviceAddr函数便可以同时取得指定设备的物理基地址和虚拟线性基地址。这个时候您便可以用这个虚拟线性基地址，再根据硬件使用说明书中的各端口寄存器的功能说明，然后使用ReadRegisterULong和WriteRegisterULong对这些端口寄存器进行 32 位模式的读写操作，即可实现设备的所有控制。

综上所述，用户使用我公司提供的驱动程序软件包将极大的方便和满足您的各种需求。但为了您更省心，别忘了在您正式阅读下面的函数说明时，先明白自己是上层用户还是底层用户，因为在《设备驱动接口函数总列表》中的备注栏里明确注明了适用对象。

另外需要申明的是，在本章和下一章中列明的关于 LabView 的接口，均属于外挂式驱动接口，他是通过 LabView 的 Call Labrary Function 功能模板实现的。它的特点是除了自身的语法略有不同以外，每一个基于 LabView 的驱动图标与 Visual C++、Visual Basic、Delphi 等语言中每个驱动函数是一一对应的，其调用流程和功能是完全相同的。那么相对于外挂式驱动接口的另一种方式是内嵌式驱动。这种驱动是完全作为 LabView 编程环境中的紧密耦合的一部分，它可以直接从 LabView 的 Functions 模板中取得，如下图所示。此种方式更适合上层用户的需要，它的最大特点是方便、快捷、简单，而且可以取得它的在线帮助。关于 LabView 的外挂式驱动和内嵌式驱动更详细的叙述，请参考 LabView 的相关演示。



LabView 内嵌式驱动接口的获取方法

第一节、设备驱动接口函数总列表（每个函数省略了前缀“PXI9100_”）

函数名	函数功能	备注
① 设备对象操作函数		
CreateDevice	创建 PXI 设备对象(用设备逻辑号)	上层及底层用户
GetDeviceCount	取得同一种 PXI 设备的总台数	上层及底层用户
ListDeviceDlg	列表所有同一种 PXI 设备的各种配置	上层及底层用户
CreateDeviceEx	用物理号创建设备对象	上层及底层用户
GetDeviceCurrentID		上层及底层用户
ReleaseDevice	关闭设备，且释放 PXI 总线设备对象	上层及底层用户
② DA 数据读取函数		
SetDevTrigLevelDA	设置触发电平(mV)	上层及底层用户
SetDevFrequencyDA	在 DA 转换过程中，动态改变采样频率	上层及底层用户
ReadSegmentInfo	设置 DA 的输出范围(只适用于 PXI9100)	上层及底层用户
InitDeviceDA	初始化设备，当返回 TRUE 后，设备即准备就绪.	上层及底层用户
WriteDeviceBulkDA	在 DA 输出前，用此函数将 DA 数据写入板载 RAM 中(程序方式)	上层及底层用户
ReadDeviceBulkDA	用此函数将板载 RAM 中的数据读回计算机(程序方式)	上层及底层用户
EnableDeviceDA	在初始化之后，使能设备	上层及底层用户
SetDeviceTrigDA	当设备使能允许后，产生软件触发事件（只有触发源为软件触发时有效）	上层及底层用户
WriteDeviceOneDA	设备对象句柄	上层及底层用户
GetDevStatusDA	在 DA 采样过程中取得设备的各种状态,返回值表示函数是否成功	上层及底层用户
DisableDeviceDA	在使设备之后，禁止设备	上层及底层用户
ReleaseDeviceDA	禁止 DA 设备，且释放资源	上层及底层用户

③ DA 校准		
StartCalibration	启动 DA 校准	
GetDACalibration	设备 DA 校准	
SetDACalibration	设备 DA 校准	
StopCalibration	停止 DA 校准	
④ DA 的硬件参数操作函数		
LoadParaDA	从 Windows 系统中读入硬件参数	
SaveParaDA	往 Windows 系统写入设备硬件参数	
ResetParaDA	将注册表中的 DA 参数恢复至出厂默认值	

使用需知：

Visual C++：

要使用如下函数关键的问题是：

首先，必须在您的源程序中包含如下语句：

#include “C:\PXI\PXI9100\INCLUDE\PXI9100.H”

注：以上语句采用默认路径和默认板号，应根据您的板号和安装情况确定 PXI9100.H 文件的正确路径，当然也可以把此文件拷到您的源程序目录中。然后加入如下语句：

#include “PXI9100.H”

另外，要在 VB 环境中用子线程以实现高速、连续数据采集与存盘，请务必使用 VB5.0 版本。当然如果您有 VB6.0 的最新版，也可以实现子线程操作。

Visual Basic:

要使用如下函数一个关键的问题是首先必须将我们提供的模块文件(*.Bas)加入到您的 VB 工程中。其方法是选择 VB 编程环境中的工程(Project)菜单，执行其中的"添加模块"(Add Module)命令，在弹出的对话中选择 PXI9100.Bas 模块文件，该文件的路径为用户安装驱动程序后其子目录 Samples\VB 下面。

请注意，因考虑 Visual C++和 Visual Basic 两种语言的兼容问题，在下列函数说明和示范程序中，所举的 Visual Basic 程序均是需要编译后在独立环境中运行。所以用户若在解释环境中运行这些代码，我们不能保证完全顺利运行。

LabVIEW/CVI :

LabVIEW 是美国国家仪器公司(National Instrument)推出的一种基于图形开发、调试和运行程序的集成化环境，是目前国际上唯一的编译型的图形化编程语言。在以 PC 机为基础的测量和工控软件中，LabVIEW 的市场普及率仅次于 C++/C 语言。LabVIEW 开发环境具有一系列优点，从其流程图式的编程、不需预先编译就存在的语法检查、调试过程使用的数据探针，到其丰富的函数功能、数值分析、信号处理和设备驱动等功能，都令人称道。关于 LabView/CVI 的进一步介绍请见本文最后一部分关于 LabView 的专述。其驱动程序接口单元模块的使用方法如下：

一、在 LabView 中打开 PXI9100.VI 文件，用鼠标单击接口单元图标，比如 CreateDevice 图标 然后按 Ctrl+C 或选择 LabView 菜单 Edit 中的 Copy 命令，接着进入用户的应用程序 LabView 中，按 Ctrl+V 或选择 LabView 菜单 Edit 中的 Paste 命令，即可将接口单元加入到用户工程中，然后按以下函数原型说明或演示程序的说明连接该接口模块即可顺利使用。

二、根据 LabView 语言本身的规定，接口单元图标以黑色的较粗的中间线为中心，以左边的方格为数据输入端，右边的方格为数据的输出端，设备对象句柄、用户分配的数据缓冲区、要求采集的数据长度等信息从接口单元左边输入端进入单元，待单元接口被执行后，需要返回给用户的数据从接口单元右边的输出端输出，其他接口完全同理。

三、在单元接口图标中，凡标有“I32”为有符号长整型 32 位数据类型，“U16”为无符号短整型 16 位数据类型，“［U16］”为无符号 16 位短整型数组或缓冲区或指针，“［U32］”与“［U16］”同理，只是位数不一样。

第二节、设备对象管理函数原型说明

♦ **创建设备对象函数（逻辑号）**

函数原型：

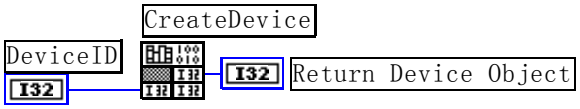
Visual C++：

HANDLE CreateDevice (int DeviceLgcID = 0)

Visual Basic：

Declare Function CreateDevice Lib "PXI9100" (Optional ByVal DeviceLgcID As Integer = 0) As Long

LabVIEW：



功能：该函数使用逻辑号创建设备对象，并返回其设备对象句柄 hDevice。只有成功获取 hDevice，您才能实现对该设备所有功能的访问。

参数：

DeviceLgcID 逻辑设备 ID(Logic Device Identifier)标识号。当向同一个 Windows 系统中加入若干相同类型的 PXI 设备时，我们的驱动程序将以该设备的“基本名称”与 DeviceLgcID 标识值为后缀的标识符来确认和管理该设备。比如若用户往 Windows 系统中加入第一个 PXI9100 模板时，驱动程序逻辑号为“0”来确认和管理第一个设备，若用户接着再添加第二个 PXI9100 模板时，则系统将以逻辑号“1”来确认和管理第二个设备，若再添加，则以此类推。所以当用户要创建设备句柄管理和操作第一个 PXI 设备时，DeviceLgcID 应置 0，第二个应置 1，也以此类推。但默认值为 0。该参数之所以称为逻辑设备号，是因为每个设备的逻辑号是不能事先由用户硬性确定的，而是由 BIOS 和操作系统加载设备时，依据主板总线编号等信息进行这个设备 ID 号分配，说得简单点，就是加载设备的顺序编号，编号的递增顺序为 0、1、2、3……。所以用户无法直接固定某一个设备的在设备列表中的物理位置，若想固定，则必须须使用物理 ID 号。

返回值：如果执行成功，则返回设备对象句柄；如果没有成功，则返回错误码 INVALID_HANDLE_VALUE。由于此函数已带容错处理，即若出错，它会自动弹出一个对话框告诉您出错的原因。您只需要对此函数的返回值作一个条件处理即可，别的任何事情您都不必做。

相关函数： CreateDevice　GetDeviceCount　ListDeviceDlg　ListDeviceDlgEx　ReleaseDevice

Visual C++程序举例

```
    :
HANDLE hDevice;     // 定义设备对象句柄
int DeviceLgcID = 0;
hDevice = CreateDevice ( DeviceLgcID ); // 创建设备对象,并取得设备对象句柄
if(hDevice == INVALIDE_HANDLE_VALUE); // 判断设备对象句柄是否有效
{
    return;     // 退出该函数
}
    :
```

Visual Basic 程序举例

```
    :
    Dim hDevice As Long   ' 定义设备对象句柄
Dim DeviceLgcID As Long
DeviceLgcID = 0
hDevice = CreateDevice ( DeviceLgcID )   ' 创建设备对象,并取得设备对象句柄
If hDevice = INVALID_HANDLE_VALUE Then   ' 判断设备对象句柄是否有效
    MsgBox “创建设备对象失败”
    Exit Sub      ' 退出该过程
End If
    :
```

- **取得本计算机系统中 PXI9100 设备的总数量**

函数原型：

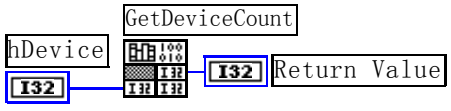
Visual C++:

int GetDeviceCount (HANDLE hDevice)

Visual Basic:

Declare Function GetDeviceCount Lib "PXI9100" (ByVal hDevice As Long) As Integer

LabVIEW：



功能：取得 PXI9100 设备的数量。
参数：hDevice设备对象句柄，它应由CreateDevice创建。
返回值：返回系统中 PXI9100 的数量。
相关函数： CreateDevice GetDeviceCount ListDeviceDlg ListDeviceDlgEx ReleaseDevice

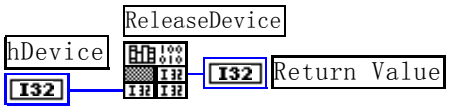
- **用对话框控件列表计算机系统中所有 PXI9100 设备各种配置信息**

函数原型：
Visual C++:
BOOL ListDeviceDlg (HANDLE hDevice)
Visual Basic:
Declare Function ListDeviceDlg Lib "PXI9100" (ByVal hDevice As Long) As Boolean
LabVIEW：
请参考相关演示程序。

功能：列表系统中 PXI9100 的硬件配置信息。
参数：hDevice设备对象句柄，它应由CreateDevice创建。
返回值：若成功，则弹出对话框控件列表所有 PXI9100 设备的配置情况。
相关函数： CreateDevice ReleaseDevice

- **释放设备对象所占的系统资源及设备对象**

函数原型：
Visual C++:
BOOL ReleaseDevice(HANDLE hDevice)
Visual Basic:
Declare Function ReleaseDevice Lib "PXI9100" (ByVal hDevice As Long) As Boolean
LabVIEW：



功能：释放设备对象所占用的系统资源及设备对象自身。
参数：hDevice设备对象句柄，它应由CreateDevice创建。
返回值：若成功，则返回TRUE， 否则返回FALSE， 用户可以用GetLastErrorEx捕获错误码。
相关函数： CreateDevice

应注意的是，CreateDevice必须和ReleaseDevice函数一一对应，即当您执行了一次CreateDevice后，再一次执行这些函数前，必须执行一次ReleaseDevice函数，以释放由CreateDevice占用的系统软硬件资源，如DMA控制器、系统内存等。只有这样，当您再次调用CreateDevice函数时，那些软硬件资源才可被再次使用。

第三节、DA 数据读取函数原型说明

- **复位整个 DA 设备状态**

函数原型：
Visual C++:
BOOL SetDevTrigLevelDA (HANDLE hDevice, _
float fTrigLevelVolt)
Visual Basic:
Declare Function SetDevTrigLevelDA Lib "PXI9100" (ByVal hDevice as Long,_
ByVal fTrigLevelVolt as Long,_
LabVIEW：
请参考演示源程序。

功能：设置触发电平(mV)。
参数：
hDevice 设备对象句柄，它应由CreateDevice创建。

pDAPara 设备对象参数结构，它决定了设备对象的各种状态及工作方式，如采样频率等。关于具体操作请参考《DA硬件参数结构》。

返回值：如果调用成功，则返回 TRUE，否则返回 FALSE，用户可用 GetLastError 捕获当前错误码，并加以分析。

相关函数： SetDevTrigLevelDA SetDevFrequencyDA ReadSegmentInfo InitDeviceDA WriteDeviceBulkDA ReadDeviceBulkDA EnableDeviceDA SetDeviceTrigDA WriteDeviceOneDA GetDevStatusDA DisableDeviceDA ReleaseDeviceDA

♦ **动态改变采样频率**

函数原型：

Visual C++:

```
BOOL SetDeviceFreqDA (HANDLE hDevice,
                      LONG nFrequency,
                      int n DAChannel)
```

Visual Basic:

```
Declare Function SetDeviceFreqDA Lib "PXI9100" (ByVal hDevice as Long, _
                                                 ByVal nFrequency As Long,_
                                                 ByVal nDAChannel As Integer) As Boolean
```

LabVIEW：

请参考演示源程序。

功能：在 DA 采样过程中，可动态改变采样频率。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

nFrequency DA 采样频率，单位为 Hz。

nNDAChannel DA 通道号，取值为[0, 1]。

返回值：如果调用成功，则返回TRUE，否则返回FALSE，用户可用GetLastError捕获当前错误码，并加以分析。

相关函数： SetDevTrigLevelDA SetDevFrequencyDA ReadSegmentInfo InitDeviceDA WriteDeviceBulkDA ReadDeviceBulkDA EnableDeviceDA SetDeviceTrigDA WriteDeviceOneDA GetDevStatusDA DisableDeviceDA ReleaseDeviceDA

♦**设置 DA 的输出范围**

函数原型：

Visual C++:

```
BOOL ReadSegmentInfo   (HANDLE hDevice,
                        LONG SegmentCount,
                        PXI9100_SEGMENT_INFO SegmentInfo[]
                        int nDAChannel);)
```

Visual Basic:

```
Declare Function ReadSegmentInfo Lib "PXI9100" (ByVal hDevice as Long, _
                                                 ByVal _SEGMENT_INFO SegmentInfo[]as Long, _
                                                 ByVal SegmentCount as Long, _
                                                 ByVal nDAChannel As Integer) As Boolean
```

LabVIEW：

请参考演示源程序。

功能：在 DA 采样过程中，设置 DA 的输出范围。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

返回值：如果调用成功，则返回TRUE，否则返回FALSE，用户可用GetLastError捕获当前错误码，并加以分析。

相关函数： SetDevTrigLevelDA SetDevFrequencyDA ReadSegmentInfo InitDeviceDA WriteDeviceBulkDA ReadDeviceBulkDA EnableDeviceDA SetDeviceTrigDA WriteDeviceOneDA GetDevStatusDA DisableDeviceDA ReleaseDeviceDA

♦ **初始化设备对象**

函数原型：

Visual C++:

```
BOOL InitDeviceDA (HANDLE hDevice,
                   LONG SegmentCount,
                   PXI9100_SEGMENT_INFO SegmentInfo[]
                   PPXI9100_PARA_DA pDAPara,
                   int nDAChannel)
```

Visual Basic:

```
Declare Function InitDeviceDA Lib "PXI9100" (ByVal hDevice As Long, _
                                             ByRef ClockSource As Long,_
                                             ByRef _SEGMENT_INFO SegmentInfo[] As Long,_
                                             ByRef _PARA_DA pDAPara As Long,_
                                             ByVal nDAChannel As Integer) As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：它负责初始化设备对象中的DA部件，为设备操作就绪有关工作做准备，如预置DA采集通道、采样频率等。但它并不启动DA设备，若要启动DA设备，须在调用此函数之后再调用StartDeviceProDA (但DA要实际输出波形，则一般要等待某种触发事件的到来)。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

SegmentCount 总工作段数[1, 666]

nDAChannel 通道号，取值为[0, 1]。

PXI9100_SEGMENT_INFO SegmentInfo[] 段信息集合

PPXI9100_PARA_DA pDAPara 硬件参数, 它仅在此函数中决定硬件状态

返回值：如果初始化设备对象成功，则返回TRUE，否则返回FALSE，用户可用GetLastError捕获当前错误码，并加以分析。

相关函数：SetDevTrigLevelDA SetDevFrequencyDA ReadSegmentInfo InitDeviceDA WriteDeviceBulkDA ReadDeviceBulkDA EnableDeviceDA SetDeviceTrigDA WriteDeviceOneDA GetDevStatusDA DisableDeviceDA ReleaseDeviceDA

- **DA 输出前，用此函数将 DA 数据写入板载 RAM 中(程序方式)**

函数原型：

Visual C++:

```
BOOL WriteDeviceBulkDA (HANDLE hDevice,
                        DABuffer[],
                        nWriteSizeWords,
                        nRetSizeWords,
                        int nDAChannel)
```

Visual Basic:

```
Declare Function WriteDeviceBulkDA Lib "PXI9100" (ByVal hDevice As Long,_
                                                  ByVal DABuffer[] As Long,_
                                                  ByValnWriteSize As Long,_
                                                  ByVal RetSizeAs Long,_
                                                  ByVal nDAChannel As Integer) As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：DA 输出前，用此函数将 DA 数据写入板载 RAM 中(程序方式)

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

DABuffer[] 携带原始 DA 数据的用户缓冲区

nWriteSizeWords 写入的数据长度(字)

nRetSizeWords 返回实际写出的长度(字)

nDAChannel 设备通道号，取值范围为[0, 1]。

返回值：如果调用成功，则返回 TRUE，且 DA 准备就绪，等待触发事件的到来就开始实际的 DA 输出， 否则返回 FALSE，用户可用 GetLastError 捕获当前错误码，并加以分析。

相关函数：SetDevTrigLevelDA SetDevFrequencyDA ReadSegmentInfo InitDeviceDA WriteDeviceBulkDA ReadDeviceBulkDA EnableDeviceDA SetDeviceTrigDA WriteDeviceOneDA GetDevStatusDA DisableDeviceDA ReleaseDeviceDA

♦ **板载 RAM 中的数据读回计算机(程序方式)**

函数原型：

Visual C++:

```
BOOL ReadDeviceBulkDA (HANDLE hDevice,
                       SHORT DABuffer[],
                       LONG nReadSizeWords,
                       PLONG nRetSizeWords
                       int nDAChannel)
```

Visual Basic:

```
Declare Function ReadDeviceBulkDALib "PXI9100" (ByVal hDevice as Long,_
                                                ByVal DABuffer[] as Long,_
                                                ByVal nReadSize as Long,_
                                                ByVal nRetSize as Long,_
                                                ByVal nDAChannel As Integer) As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：用此函数将板载 RAM 中的数据读回计算机(程序方式)。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

SHORT DABuffer[] DA 数据的用户缓冲区

nReadSizeWords 读入的数据长度(字)

nRetSizeWords 返回实际读出的长度(字)

nDAChannel 设备通道号，取值范围为[0, 1]。

返回值：如果调用成功，则返回 TRUE，且 DA 立刻停止转换，否则返回 FALSE，用户可用 GetLastError 捕获当前错误码，并加以分析。

相关函数：SetDevTrigLevelDA SetDevFrequencyDA ReadSegmentInfo InitDeviceDA WriteDeviceBulkDA ReadDeviceBulkDA EnableDeviceDA SetDeviceTrigDA WriteDeviceOneDA GetDevStatusDA DisableDeviceDA ReleaseDeviceDA

♦**初始化之后，使能设备取得**

函数原型：

Visual C++:

```
BOOL EnableDeviceDA  (HANDLE hDevice,
                      int nDAChannel)
```

Visual Basic:

```
Declare Function GetDeviceStatusProDA Lib "PXI9100" (ByVal hDevice As Long,_
                                                     ByVal nDAChannel ) As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：在初始化之后，使能设备。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

nDAChanne DA 通道号[0, 1]。

返回值：如果调用成功，则返回 TRUE，否则返回 FALSE，用户可用 GetLastError 捕获当前错误码，并加以分析。

相关函数：SetDevTrigLevelDA SetDevFrequencyDA ReadSegmentInfo InitDeviceDA

WriteDeviceBulkDA ReadDeviceBulkDA EnableDeviceDA SetDeviceTrigDA
WriteDeviceOneDA GetDevStatusDA DisableDeviceDA ReleaseDeviceDA

♦设备使能允许后，产生软件触发事件（只有触发源为软件触发时有效）

函数原型：

Visual C++I:

```
BOOL WriteDeviceProDA (HANDLE hDevice,
                       BOOL bSetSyncTrig,
                       int nDAChannel)
```

Visual Basic:

```
Declare Function WriteDeviceProDA Lib "PXI9100" (ByVal hDevice As Long,_
                                                 ByRef bSetSyncTrig As Long,_
                                                 ByVal nDAChannel As Integer) As Boolean
```

LabVIEW：

请参考相关演示程序。

功能：当设备使能允许后，产生软件触发事件（只有触发源为软件触发时有效）。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

bSetSyncTrig 是否置同步触发。

nDAChannel 设备通道号，取值范围为[0, 1]。

返回值：如果调用成功，则返回 TRUE，且 DA 立刻停止转换，否则返回 FALSE，用户可用 GetLastError 捕获当前错误码，并加以分析。

相关函数：SetDevTrigLevelDA SetDevFrequencyDA ReadSegmentInfo InitDeviceDA
WriteDeviceBulkDA ReadDeviceBulkDA EnableDeviceDA SetDeviceTrigDA
WriteDeviceOneDA GetDevStatusDA DisableDeviceDA ReleaseDeviceDA

♦ **DA 采样过程中取得设备的各种状态**

函数原型：

Visual C++:

```
BOOL GetDevStatusDA (HANDLE hDevice,
                     PPXI9100_STATUS_DA pDAStatus
                     int nDAChannel)
```

Visual Basic:

```
Declare Function GetDevStatusDA Lib "PXI9100" (ByVal hDevice as Long, _
                                               ByVal _STATUS_DA pDAStatus as Long, _
                                               ByVal nDAChannel As Integer) As Boolean
```

LabView：

请参考相关演示程序。

功能：在 DA 采样过程中取得设备的各种状态,返回值表示函数是否成功。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

PPXI9100_STATUS_DA pDAStatus DA 的各种信息结构体。

nDAChannel 设备通道号，取值范围为[0, 1]。

返回值：如果调用成功，则返回 TRUE，且 DA 立刻停止转换，否则返回 FALSE，用户可用 GetLastError 捕获当前错误码，并加以分析。

相关函数：SetDevTrigLevelDA SetDevFrequencyDA ReadSegmentInfo InitDeviceDA
WriteDeviceBulkDA ReadDeviceBulkDA EnableDeviceDA SetDeviceTrigDA
WriteDeviceOneDA GetDevStatusDA DisableDeviceDA ReleaseDeviceDA

♦ **使设备之后，禁止设备**

函数原型：

Visual C++:

```
BOOL DisableDeviceDA (HANDLE hDevice,
                      int nDAChannel)
```

Visual Basic:

```
Declare Function DisableDeviceDA Lib "PXI9100" (ByVal hDevice as Long, _
```

ByVal nDAChannel As Integer) As Boolean

LabView：

请参考相关演示程序。

功能：在使设备之后，禁止设备。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

PPXI9100_STATUS_DA pDAStatus DA 的各种信息结构体。

nDAChannel 设备通道号，取值范围为[0, 1]。

返回值：如果调用成功，则返回 TRUE，且 DA 立刻停止转换，否则返回 FALSE，用户可用 GetLastError 捕获当前错误码，并加以分析。

相关函数：SetDevTrigLevelDA SetDevFrequencyDA ReadSegmentInfo InitDeviceDA WriteDeviceBulkDA ReadDeviceBulkDA EnableDeviceDA SetDeviceTrigDA WriteDeviceOneDA GetDevStatusDA DisableDeviceDA ReleaseDeviceDA

♦ 禁止 DA 设备，且释放资源

函数原型：

Visual C++:

```
BOOL ReleaseDeviceDA (HANDLE hDevice,
                      PPXI9100_STATUS_DA pDAStatus
                      int nDAChannel)
```

Visual Basic:

```
Declare Function ReleaseDeviceDA Lib "PXI9100" (ByVal hDevice as Long, _
                                                ByVal _STATUS_DA pDAStatus as Long, _
                                                ByVal nDAChannel As Integer) As Boolean
```

LabView：

请参考相关演示程序。

功能：在使设备之后，禁止设备。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

PPXI9100_STATUS_DA pDAStatus DA 的各种信息结构体。

nDAChannel 设备通道号，取值范围为[0, 1]。

返回值：如果调用成功，则返回 TRUE，且 DA 立刻停止转换，否则返回 FALSE，用户可用 GetLastError 捕获当前错误码，并加以分析。

相关函数：SetDevTrigLevelDA SetDevFrequencyDA ReadSegmentInfo InitDeviceDA WriteDeviceBulkDA ReadDeviceBulkDA EnableDeviceDA SetDeviceTrigDA WriteDeviceOneDA GetDevStatusDA DisableDeviceDA ReleaseDeviceDA

♦ DA 单点输出

函数原型：

Visual C++:

```
BOOL GetDevStatusDA (HANDLE hDevice,
                     ULON GulDataCode,
                     int nDAChannel)
```

Visual Basic:

```
Declare Function GetDevStatusDA Lib "PXI9100" (ByVal hDevice as Long, _
                                               ByVal GulDataCode as Long, _
                                               ByVal nDAChannel As Integer) As Boolean
```

LabView：

请参考相关演示程序。

功能：DA 单点输出。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

ulDataCode 输入码值（0—4095）。

nDAChannel 设备通道号，取值范围为[0, 1]。

返回值：如果调用成功，则返回 TRUE，且 DA 立刻停止转换，否则返回 FALSE，用户可用 GetLastError 捕获当前错误码，并加以分析。

相关函数：SetDevTrigLevelDA SetDevFrequencyDA ReadSegmentInfo InitDeviceDA WriteDeviceBulkDA ReadDeviceBulkDA EnableDeviceDA SetDeviceTrigDA WriteDeviceOneDA GetDevStatusDA DisableDeviceDA ReleaseDeviceDA

第四节、DA 校准

♦启动 DA 校准

函数原型：

Visual C++:

```
BOOL StartCalibration    (HANDLE hDevice,)
```

Visual Basic:

```
Declare Function GetDevStatusDA Lib "PXI9100" (ByVal hDevice as Long, _ ) As Boolean
```

LabView：

请参考相关演示程序。

功能：启动 DA 校准。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

返回值：如果调用成功，则返回 TRUE，且 DA 立刻停止转换，否则返回 FALSE，用户可用 GetLastError 捕获当前错误码，并加以分析。

相关函数：StartCalibration GetDACalibration SetDACalibration StopCalibration

♦ DA 设备校准

函数原型：

Visual C++:

```
BOOL GetDACalibration (HANDLE hDevice,
                       LONG OutputRange,
                       LONG CalMode,
                       PLONG pCalData,
                       int nDAChannel)
```

Visual Basic:

```
Declare Function GetDACalibration Lib "PXI9100" (ByVal hDevice as Long, _
                                                 ByVal OutputRange as Long, _
                                                 ByVal CalMode as Long, _
                                                 ByVal pCalData pDAStatus as Long, _
                                                 ByVal nDAChannel As Integer) As Boolean
```

LabView：

请参考相关演示程序。

功能：设备 DA 校准。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

OutputRange 输出量程,分别控制两个通道。

CalMode 0 为零点校准，1 为满度校准

PCalData 校准值

nDAChannel 设备通道号，取值范围为[0, 1]。

返回值：如果调用成功，则返回 TRUE，且 DA 立刻停止转换，否则返回 FALSE，用户可用 GetLastError 捕获当前错误码，并加以分析。

相关函数: StartCalibration GetDACalibration SetDACalibration StopCalibration

♦ DA 设备校准

函数原型：

Visual C++:

```
BOOL SetDACalibration (HANDLE hDevice,
                       LONG OutputRange,
                       LONG CalMode,
                       LONG pCalData,
                       int nDAChannel)
```

Visual Basic:

```
Declare Function SetDACalibrationLib "PXI9100" (ByVal hDevice as Long, _
                                                 ByVal OutputRange as Long, _
                                                 ByVal CalMode as Long, _
                                                 ByVal CalData pDAStatus as Long, _
                                                 ByVal nDAChannel As Integer) As Boolean
```

LabView：

请参考相关演示程序。

功能：设备 DA 校准。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

OutputRange 输出量程,分别控制两个通道。

CalMode 0 为零点校准，1 为满度校准

PCalData 校准值

nDAChannel 设备通道号，取值范围为[0, 1]。

返回值：如果调用成功，则返回 TRUE，且 DA 立刻停止转换，否则返回 FALSE，用户可用 GetLastError 捕获当前错误码，并加以分析。

相关函数：StartCalibration GetDACalibration SetDACalibration StopCalibration

- **停止 DA 校准**

函数原型：

Visual C++:

```
BOOL StopCalibration (HANDLE hDevice,)
```

Visual Basic:

```
Declare Function StopCalibration Lib "PXI9100" (ByVal hDevice as Long, _)
```

LabView：

请参考相关演示程序。

功能：停止 DA 校准。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

返回值：如果调用成功，则返回 TRUE，且 DA 立刻停止转换，否则返回 FALSE，用户可用 GetLastError 捕获当前错误码，并加以分析。

相关函数：StartCalibration GetDACalibration SetDACalibration StopCalibration

第四章 硬件参数结构

各段信息（PXI9100_PARA_DA）

Visual C++：

```
typedef struct _PXI9100_SEGMENT_INFO
{
    LONG SegLoopCount;      // 每个段在大循环中的小循环次数,取值为[1, 16777215]
    LONG SegmentSize;       // 每个段在 RAM 中的长度(单位：字/点)
```

```
} PXI9100_SEGMENT_INFO, *PPXI9100_SEGMENT_INFO;
```

Visual Basic：

```
Type PXI9100_PARA_DA
    SegLoopCount As Long
    SegmentSize As Long
End Type
```

LabVIEW:

请参考相关演示程序。

DA 的工作参数

```
typedef struct _PXI9100_PARA_DA
{
    LONG OutputRange;          // 输出量程
    LONG Frequency;            // 点频率[0.010Hz, 80MHz], 为正数时单位 Hz
    LONG LoopCount;            // 整个 RAM 的大循环次数,=0:无限循环, =n:表示 n 次循环(1<n<32768)
    LONG TriggerMode;          // 触发模式选择
    LONG TriggerSource;        // 触发源选择
    LONG TriggerDir;       // 触发方向选择
    LONG bSingleOut;       // 是否单点输出
    LONG ClockSource;          // 时钟源选择
} PXI9100_PARA_DA, *PPXI9100_PARA_DA;
```

Visual Basic：

```
Type PXI9100_PARA_DA
    OutputRange As Long
    Frequency As Long
    LoopCount As Long
    TriggerMode As Long
    TriggerSource As Long
    TriggerDir As Long
    bSingleOut As Long
    ClockSource As Long
End Type
```

LabVIEW:

请参考相关演示程序。

此结构主要用于设定设备DA硬件参数值，用这个参数结构对设备进行硬件配置完全由InitDeviceProDA自动完成。用户只需要对这个结构体中的各成员简单赋值即可。

OutputRange 模拟量输出范围参数方式选择。

常量名	常量值	功能定义
PXI9100_OUTPUT_FRE_LOW	0x00	低速模式
PXI9100_OUTPUT_FRE_HIGH	0x01	高速模式

OutMode 模拟量输出选项。

常量名	常量值	功能定义
PXI9100_OUTPUT_N5000_P5000mV	0x00	±5000mV
PXI9100_OUTPUT_N10000_P10000mV	0x01	±10000mV

TriggerSource 变量触发源选项。

常量名	常量值	功能定义
PXI9100_TRIGSRC_SOFT_DA	0x0000	软件触发
PXI9100_TRIGSRC_ATR_DA	0x00001	ATR 硬件模拟触发

TriggerMode 成员变量触发模式选项。

常量名	常量值	功能定义
PXI9100_TRIGMODE_SINGLE	0x00	单次触发
PXI9100_TRIGMODE_CONTINUOUS	0x01	连续触发
PXI9100_TRIGMODE_STEPED	0x02	单步触发
PXI9100_TRIGMODE_BURST	0x03	紧急触发

TriggerDir 成员触发方向选项。

常量名	常量值	功能定义
PXI9100_TRIGDIR_NEGATIVE	0x00	负向触发(低脉冲/下降沿触发)
PXI9100_TRIGDIR_POSITIVE	0x01	正向触发(高脉冲/上升沿触发)
PXI9100_TRIGDIR_POSIT_NEGAT	0x02	正负向触发(高/低脉冲或上升/下降沿触发)

ClockSource 时钟源所使用的选项。

常量名	常量值	功能定义
PXI9100_CLOCKSRC_IN	0x00	内部时钟
PXI9100_CLOCKSRC_OUT	0x01	外部时钟

各段的头信息在 RAM 中的地址空间。

常量名	常量值	功能定义
PXI9100_SEGMENT_HEADER_SIZE	0x0C	段头信息的长度

CreateFileObject 所用的文件操作方式控制字。

常量名	常量值	功能定义
PXI9100_modeRead	0x00	只读文件方式
PXI9100_modeWrite	0x01	只写文件方式
PXI9100_modeReadWrite	0x02	既读又写文件方式
PXI9100_modeCreate	0x03	如果文件不存可以创建该文件，如果存在，则重建此文件，并清 0
PXI9100_typeText	0x04	以文本方式操作文件

RegisterID 所使用的选项(有效部分)。

常量名	常量值	功能定义
PXI9100_REG_RAM_CPLD	0x00	0 号寄存器对应配置所有相关寄存器使用(使用 LinearAddr)
PXI9100_REG_RAM_DA0BUFFER	0x01	1 号寄存器对应板上 DA 缓冲区所使用的内存模式基地址(使用 LinearAddr)
PXI9100_REG_RAM_DA1BUFFER	0x02	2 号寄存器对应板上 DA 缓冲区所使用的内存模式基地址(使用 LinearAddr)

DA采样的实际硬件参数

```
typedef struct _PXI9100_STATUS_DA
{
    LONG bEnable;          // DA使能启动标志，＝TRUE表示DA已被使能, = FALSE表示DA被禁止
    LONG bTrigFlag;        // 触发标志是否有效，=TRUE表示触点标有效， = FALSE表示无效（即触发点未到）
    LONG bConverting;      // DA是否正在转换， =TRUE:表示正在转换， = FALS表示转换完成
    LONG nCurSegNum;       // 可读取的RAM段号，取值为[0，SegmentCount-1]，(注SegmentCount为InitDeviceDA函数的参数)
    LONG nCurSegAddr;      // 可读取的RAM段地址
```

```
    LONG nCurLoopCount; //  当前总循环次数
    LONG nCurSegLoopCount; //  当前段循环次数
} PXI9100_STATUS_DA, *PPXI9100_STATUS_DA;
```

第五章 数据格式转换与排列规则

第一节、AD 原码 LSB 数据转换成电压值的换算方法

首先应根据设备实际位数屏蔽掉不用的高位，然后依其所选量程，按照下表公式进行换算即可。这里只以缓冲区 ADBuffer[]中的第 1 个点 ADBuffer[0]为例。

量程(mV)	计算机语言换算公式(ANSI C 语法)	Volt 取值范围（mV）
±5000mV	Volt = (10000.00/4096) * (ADBuffer[0]&0xFFF) – 5000.00	[-5000, +4997.55]
±1000mV	Volt = (2000.00/4096) * (ADBuffer[0]&0xFFF) – 1000.00	[-1000, +999.51]

下面举例说明各种语言的换算过程（以±5000mV 量程为例）

Visual C++：

```
Lsb = ADBuffer[0]&0xFFF;
    Volt = (10000.00/4096) * Lsb -5000.00;
```

Visual Basic：

```
Lsb = ADBuffer [0] And &HFFF
Volt = (10000.00/4096) * Lsb – 5000.00
```

LabVIEW:

请参考相关演示程序。

第二节、AD 采集函数的 ADBuffer 缓冲区中的数据排放规则

数据缓冲区索引号	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	…
通道号	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	…

第三节、AD 测试应用程序创建并形成的数据文件格式

首先该数据文件从始端 0 字节位置开始往后至第 HeadSizeBytes 字节位置宽度属于文件头信息，而从 HeadSizeBytes 开始才是真正的 AD 数据。HeadSizeBytes 的取值通常等于本头信息的字节数大小。文件头信息包含的内容如下结构体所示。对于更详细的内容请参考 Visual C++高级演示工程中的 UserDef.h 文件。

```
typedef struct _FILE_HEADER
{
    LONG HeadSizeBytes;                 // 文件头信息长度
    LONG FileType;                      // 该设备数据文件共有的成员
    LONG BusType;                       // 设备总线类型(DEFAULT_BUS_TYPE)
    LONG DeviceNum;                     // 该设备的编号(DEFAULT_DEVICE_NUM)
    LONG HeadVersion;                   // 头信息版本(D31-D16=Major,D15-D0=Minijor) = 1.0
    LONG VoltBottomRange;               // 量程下限(mV)
    LONG VoltTopRange;                      // 量程上限(mV)

    LONG ChannelCount;                      // 通道总数
    LONG DataWidth;                         // 设备的精度(分辩率)
    LONG bXorHighBit;                       // 是否高位求反(为 1 则求反)
    PCIe8532_PARA_AD ADPara;            // 保存硬件参数
    PCIe8532_STATUS_AD ADStatus;        // 保存硬件参数
    LONG CrystalFreq;                   // 晶振频率
```

```
    LONG ChannelNum;                    // 通道号
    LONG HeadEndFlag;                   // 头信息结束位
} FILE_HEADER，*PFILE_HEADER;
```

AD 数据的格式为 16 位二进制格式，它的排放规则与在 ADBuffer 缓冲区排放的规则一样，即每 16 位二进制(字)数据对应一个 16 位 AD 数据。您只需要先开辟一个 16 位整型数组或缓冲区，然后将磁盘数据从指定位置(即双字节对齐的某个位置)读入数组或缓冲区，然后访问数组中的每个元素，即是对相应 AD 数据的访问。

第六章 上层用户函数接口应用实例

第一节、简易程序演示说明

一、怎样使用WriteDeviceBulkDA函数进行批量DA数据输出

其详细应用实例及工程级代码请参考 Visual C++简易演示系统及源程序，您先点击 Windows 系统的[开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程(主要参考 PXI9100.h 和 Sys.cpp)。

[程序]｜[阿尔泰测控演示系统]｜[Microsoft Visual C++]｜[简易代码演示]｜[DA 批量输出演示源程序]

其简易程序默认存放路径为：系统盘\PXI\PXI9100\SAMPLES\VC\SIMPLE\DA\BULK

其他语言的演示可以用上面类似的方法找到。

第二节、高级程序演示说明

高级程序演示了本设备的所有功能，您先点击 Windows 系统的[开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程(主要参考 PXI9100.h 和 DADoc.cpp)。

[程序]｜[阿尔泰测控演示系统]｜[Microsoft Visual C++]｜[高级代码演示]

其默认存放路径为：系统盘\PXI\ PXI9100\SAMPLES\VC\ADVANCED

其他语言的演示可以用上面类似的方法找到。

第七章 共用函数介绍

这部分函数不参与本设备的实际操作，它只是为您编写数据采集与处理程序时的有力手段，使您编写应用程序更容易，使您的应用程序更高效。

第一节、公用接口函数总列表（每个函数省略了前缀“PXI9100_”）

函数名	函数功能	备注
①数字 I/O 输入输出函数		
GetDeviceDI	取得开关量状态	底层用户
SetDeviceDO	输出开关量状态	底层用户
RetDeviceDO	输出开关量状态	底层用户
②PXI 总线内存映射寄存器操作函数		
GetDeviceBar	取得指定的指定设备寄存器组 BAR 地址	底层用户
GetDeviceAddr	取得指定的指定设备寄存器组 BAR 地址	
WriteRegisterByte	以字节(8Bit)方式写寄存器端口	底层用户
WriteRegisterWord	以字(16Bit)方式写寄存器端口	底层用户
WriteRegisterULong	以双字(32Bit)方式写寄存器端口	底层用户
ReadRegisterByte	以字节(8Bit)方式读寄存器端口	底层用户
ReadRegisterWord	以字(16Bit)方式读寄存器端口	底层用户
ReadRegisterULong	以双字(32Bit)方式读寄存器端口	底层用户
③ISA 总线 I/O 端口操作函数		
WritePortByte	以字节(8Bit)方式写 I/O 端口	用户程序操作端口

WritePortWord	以字(16Bit)方式写 I/O 端口	用户程序操作端口
WritePortULong	以无符号双字(32Bit)方式写 I/O 端口	用户程序操作端口
ReadPortByte	以字节(8Bit)方式读 I/O 端口	用户程序操作端口
ReadPortWord	以字(16Bit)方式读 I/O 端口	用户程序操作端口
ReadPortULong	以无符号双字(32Bit)方式读 I/O 端口	用户程序操作端口
④线程操作函数		
CreateSystemEvent	创建系统内核事件对象	用于线程同步或中断
ReleaseSystemEvent	释放系统内核事件对象	

第二节、PXI 内存映射寄存器操作函数原型说明

- **取得开关量状态**

函数原型：

Visual C++:

```
BOOL GetDeviceDI ( HANDLE hDevice,
                   BYTE bDISts [8])
```

Visual Basic:

```
Declare Function GetDeviceDI "PXI9100" (ByVal hDevice As Long, _
                                        ByVal bDISts [8]) As Boolean
```

LabVIEW:

功能：取得开关量状态。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

BYTE bDISts [8]) 开关输入状态(注意: 必须定义为 8 个字节元素的数组)

相关函数： GetDeviceDI SetDeviceDO RetDeviceDO

- **输出开关量状态**

函数原型：

Visual C++:

```
BOOL SetDeviceDO ( HANDLE hDevice,
                   BYTE bDOSts [8])
```

Visual Basic:

```
Declare Function GetDeviceDI "PXI9100" (ByVal hDevice As Long, _
                                        ByVal bDOSts [8]) As Boolean
```

LabVIEW:

功能：输出开关量状态。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

BYTE bDOSts [8]) 开关输出状态(注意: 必须定义为 8 个字节元素的数组)

相关函数： GetDeviceDI SetDeviceDO RetDeviceDO

- **输出开关量状态**

函数原型：

Visual C++:

```
BOOL RetDeviceDO ( HANDLE hDevice,
                   BYTE bDOSts [8])
```

Visual Basic:

```
Declare Function RetDeviceDO"PXI9100" (ByVal hDevice As Long, _
                                       ByVal bDOSts [8]) As Boolean
```

LabVIEW:

功能：取得开关量状态。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

BYTE bDOSts [8]) 开关输出状态(注意: 必须定义为 8 个字节元素的数组)

相关函数： GetDeviceDI SetDeviceDO RetDeviceDO

第三节、PXI 内存映射寄存器操作函数原型说明

♦ **取得指定内存映射寄存器的线性地址和物理地址**

函数原型：

Visual C++:

```
BOOL GetDeviceBar ( HANDLE hDevice,
                    PUCHAR pbPCIBar[6])
```

Visual Basic:

```
Declare Function GetDeviceAddr Lib "PXI9100" (ByVal hDevice As Long, _
                                               ByVal pbPCIBar[6]) As Boolean
```

LabVIEW:

功能：取得指定的指定设备寄存器组 BAR 地址。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pbPCIBar[6]) 返回 PCI BAR 所有地址,具体 PCI BAR 中有多少可用地址请看硬件说明书

相关函数： CreateDevice GetDeviceAddr WriteRegisterByte
WriteRegisterWord WriteRegisterULong ReadRegisterByte
ReadRegisterWord ReadRegisterULong ReleaseDevice

♦ **取得指定的指定设备 ID 号的映射寄存器的线性基地址，返回设备总数备固件及程序版本**

函数原型：

Visual C++:

```
BOOL GetDeviceAddr ( HANDLE hDevice,
                     PUCHAR* MemCPLDBase,
                     PUCHAR* MemDA0Buffer,
                     PUCHAR* MemDA1Buffer)
```

Visual Basic:

```
Declare Function GetDevVersion Lib "PXI9100" (ByVal hDevice As Long, _
                                              ByVal MemCPLDBase As Long, _
                                              ByVal MemDA0Buffer As Long, _
                                              ByVal MemDA1Buffer) As Boolean
```

LabVIEW:

功能：取得指定的指定设备 ID 号的映射寄存器的线性基地址，返回设备总数。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

MemCPLDBase 返回指定映射寄存器的线性地址。

MemDA0Buffer 返回指定映射寄存器的线性地址。

MemDA1Buffer 返回指定映射寄存器的线性地址。

相关函数： CreateDevice GetDeviceAddr WriteRegisterByte
WriteRegisterWord WriteRegisterULong ReadRegisterByte
ReadRegisterWord ReadRegisterULong ReleaseDevice

♦ **以单字节（即 8 位）方式写 PXI 内存映射寄存器的某个单元**

函数原型：

Visual C++:

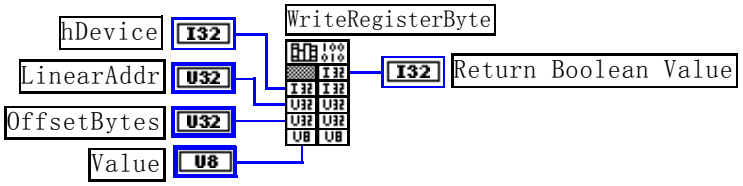
```
BOOL WriteRegisterByte( HANDLE hDevice,
                        PUCHAR pbLinearAddr,
                        ULONG OffsetBytes,
                        BYTE Value)
```

Visual Basic:

```
Declare Function WriteRegisterByte Lib "PXI9100" (ByVal hDevice As Long, _
```

```
                                        ByVal pbLinearAddr As Long, _
                                        ByVal OffsetBytes As Long, _
                                        ByVal Value As Byte ) As Boolean
```

LabVIEW:



功能：以单字节（即 8 位）方式写 PXI 内存映射寄存器。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pbLinearAddr 指定寄存器的线性基地址,它等于 GetDeviceAddr 中的 pbLinearAddr 参数返回值

OffsetBytes 相对于 LinearAddr 线性基地址的偏移字节数，它与 LinearAddr 两个参数共同确定WriteRegisterByte函数所访问的映射寄存器的内存单元。

Value 输出 8 位整数。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice　GetDeviceAddr　WriteRegisterByte
WriteRegisterWord　WriteRegisterULong　ReadRegisterByte
ReadRegisterWord　ReadRegisterULong　ReleaseDevice

Visual C++程序举例:

```
        :
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
hDevice = CreateDevice(0)
if (!GetDeviceAddr(hDevice, &LinearAddr,  &PhysAddr, 0) )
{
    AfxMessageBox "取得设备地址失败…";
}
OffsetBytes = 100;    // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
WriteRegisterByte(hDevice, LinearAddr, OffsetBytes, 0x20);  // 往指定映射寄存器单元写入 8 位的十六进制数据 20
ReleaseDevice( hDevice );  // 释放设备对象
    :
```

Visual Basic 程序举例:

```
    :
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, LinearAddr, PhysAddr, 0)
OffsetBytes = 100
WriteRegisterByte( hDevice, LinearAddr, OffsetBytes, &H20)
ReleaseDevice(hDevice)
    :
```

♦ **以双字节（即 16 位）方式写 PXI 内存映射寄存器的某个单元**

函数原型：

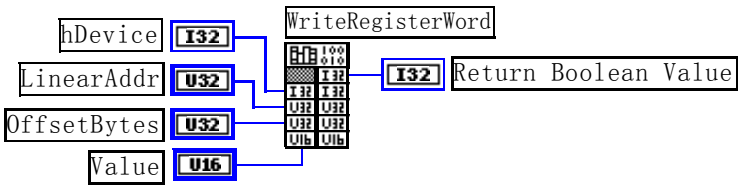
Visual C++:

```
BOOL WriteRegisterWord( HANDLE hDevice,
                        PUCHAR pbLinearAddr
                        ULONG OffsetBytes,
                        WORD Value)
```

Visual Basic:

```
Declare Function WriteRegisterWord Lib "PXI9100" (ByVal hDevice As Long, _
                                                  ByVal pbLinearAddr As Long, _
                                                  ByVal OffsetBytes As Long, _
                                                  ByVal Value As Integer) As Boolean
```

LabVIEW:



功能：以双字节（即 16 位）方式写 PXI 内存映射寄存器。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pbLinearAddr　PXI设备内存映射寄存器的线性基地址，它的值应由GetDeviceAddr确定。

OffsetBytes　相对于LinearAddr线性基地址的偏移字节数，它与LinearAddr两个参数共同确定WriteRegisterWord函数所访问的映射寄存器的内存单元。

Value　输出 16 位整型值。

返回值：无。

相关函数： CreateDevice　GetDeviceAddr　WriteRegisterByte　WriteRegisterWord　WriteRegisterULong　ReadRegisterByte　ReadRegisterWord　ReadRegisterULong　ReleaseDevice

Visual C++程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
hDevice = CreateDevice(0)
if (!GetDeviceAddr(hDevice, &LinearAddr,  &PhysAddr, 0) )
{
    AfxMessageBox "取得设备地址失败…";
}
OffsetBytes = 100;    // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
WriteRegisterWord(hDevice, LinearAddr, OffsetBytes, 0x2000);  // 往指定映射寄存器单元写入 16 位的十六进制数据
ReleaseDevice( hDevice );  // 释放设备对象
    :
```

Visual Basic 程序举例:

```
    :
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, LinearAddr, PhysAddr, 0)
OffsetBytes=100
WriteRegisterWord( hDevice, LinearAddr, OffsetBytes, &H2000)
ReleaseDevice(hDevice)
    :
```

♦ **以四字节（即 32 位）方式写 PXI 内存映射寄存器的某个单元**

函数原型：

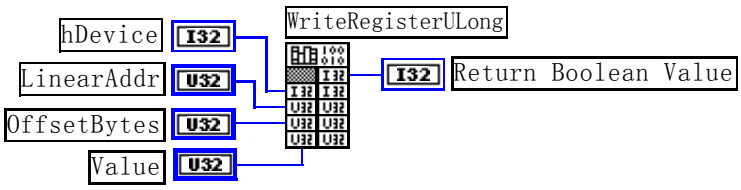
Visual C++:

```
BOOL WriteRegisterULong( HANDLE hDevice,
                         PUCHAR pbLinearAddr,
                         ULONG OffsetBytes,
                         ULONG Value)
```

Visual Basic:

```
Declare Function WriteRegisterULong Lib "PXI9100" (ByVal hDevice As Long, _
                                                   ByVal pbLinearAddr As Long, _
                                                   ByVal OffsetBytes As Long, _
                                                   ByVal Value As Long) As Boolean
```

LabVIEW:



功能：以四字节（即 32 位）方式写 PXI 内存映射寄存器。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pbLinearAddr PXI设备内存映射寄存器的线性基地址，它的值应由GetDeviceAddr确定。

OffsetBytes　相对于LinearAddr线性基地址的偏移字节数，它与LinearAddr两个参数共同确定WriteRegisterULong函数所访问的映射寄存器的内存单元。

Value 输出 32 位整型值。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice　GetDeviceAddr　WriteRegisterByte　WriteRegisterWord　WriteRegisterULong　ReadRegisterByte　ReadRegisterWord　ReadRegisterULong　ReleaseDevice

Visual C++程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
hDevice = CreateDevice(0)
if (!GetDeviceAddr(hDevice, &LinearAddr,  &PhysAddr, 0) )
{
        AfxMessageBox "取得设备地址失败…";
}
OffsetBytes=100;      // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
WriteRegisterULong(hDevice, LinearAddr, OffsetBytes, 0x20000000);  // 往指定映射寄存器单元写入 32 位的十六进制数据
ReleaseDevice( hDevice );  // 释放设备对象
    :
```

Visual Basic 程序举例:

```
    :
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, LinearAddr, PhysAddr, 0)
OffsetBytes = 100
WriteRegisterULong( hDevice, LinearAddr, OffsetBytes, &H20000000)
ReleaseDevice(hDevice)
    :
```

- **以单字节（即 8 位）方式读 PXI 内存映射寄存器的某个单元**

函数原型：

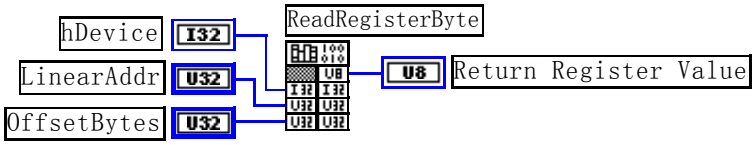
Visual C++:

```
BYTE ReadRegisterByte( HANDLE hDevice,
                       PUCHAR pbLinearAddr,
                       ULONG OffsetBytes)
```

Visual Basic:

```
Declare Function ReadRegisterByte Lib "PXI9100" (ByVal hDevice As Long, _
                                                 ByVal pbLinearAddr As Long, _
                                                 ByVal OffsetBytes As Long) As Byte
```

LabVIEW:



功能：以单字节（即 8 位）方式读 PXI 内存映射寄存器的指定单元。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pbLinearAddr PXI设备内存映射寄存器的线性基地址，它的值应由GetDeviceAddr确定。

OffsetBytes 相对于LinearAddr线性基地址的偏移字节数，它与LinearAddr两个参数共同确定ReadRegisterByte函数所访问的映射寄存器的内存单元。

返回值：返回从指定内存映射寄存器单元所读取的 8 位数据。

相关函数： CreateDevice GetDeviceAddr WriteRegisterByte
WriteRegisterWord WriteRegisterULong ReadRegisterByte
ReadRegisterWord ReadRegisterULong ReleaseDevice

Visual C++程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr,   PhysAddr, OffsetBytes;
BYTE   Value;
hDevice = CreateDevice(0);  // 创建设备对象
GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0);    // 取得 PXI 设备 0 号映射寄存器的线性基地址
OffsetBytes = 100;    // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
Value = ReadRegisterByte(hDevice, LinearAddr, OffsetBytes);  // 从指定映射寄存器单元读入 8 位数据
ReleaseDevice( hDevice );  // 释放设备对象
    :
```

Visual Basic 程序举例:

```
    :
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
Dim Value As Byte
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, Linearddr, PhysAddr, 0)
OffsetBytes = 100
Value = ReadRegisterByte( hDevice, LinearAddr, OffsetBytes)
ReleaseDevice(hDevice)
    :
```

◆ **以双字节（即 16 位）方式读 PXI 内存映射寄存器的某个单元**

函数原型：

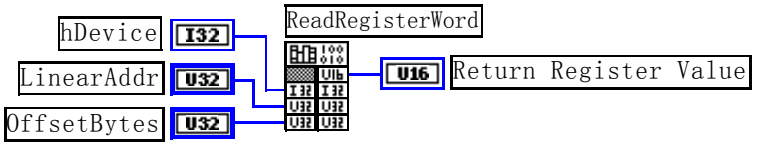
Visual C++:

```
WORD ReadRegisterWord( HANDLE hDevice,
                       PUCHAR pbLinearAddr,
                       ULONG OffsetBytes)
```

Visual Basic:

```
Declare Function ReadRegisterWord Lib "PXI9100" (ByVal hDevice As Long, _
                                                 ByVal pbLinearAddr As Long, _
                                                 ByVal OffsetBytes As Long) As Integer
```

LabVIEW:



功能：以双字节（即 16 位）方式读 PXI 内存映射寄存器的指定单元。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pbLinearAddr PXI设备内存映射寄存器的线性基地址，它的值应由GetDeviceAddr确定。

OffsetBytes 相对于LinearAddr线性基地址的偏移字节数，它与LinearAddr两个参数共同确定ReadRegisterWord函数所访问的映射寄存器的内存单元。

返回值：返回从指定内存映射寄存器单元所读取的 16 位数据。

相关函数： CreateDevice GetDeviceAddr WriteRegisterByte
WriteRegisterWord WriteRegisterULong ReadRegisterByte
ReadRegisterWord ReadRegisterULong ReleaseDevice

Visual C++程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr,   PhysAddr, OffsetBytes;
WORD Value;
hDevice = CreateDevice(0);   // 创建设备对象
GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0);    // 取得 PXI 设备 0 号映射寄存器的线性基地址
OffsetBytes = 100;    // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
Value = ReadRegisterWord(hDevice, LinearAddr, OffsetBytes);   // 从指定映射寄存器单元读入 16 位数据
ReleaseDevice( hDevice );   // 释放设备对象
    :
```

Visual Basic 程序举例:

```
    :
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
Dim Value As Word
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, Linearddr, PhysAddr, 0)
OffsetBytes = 100
Value = ReadRegisterWord( hDevice, LinearAddr, OffsetBytes)
ReleaseDevice(hDevice)
    :
```

- **以四字节（即 32 位）方式读 PXI 内存映射寄存器的某个单元**

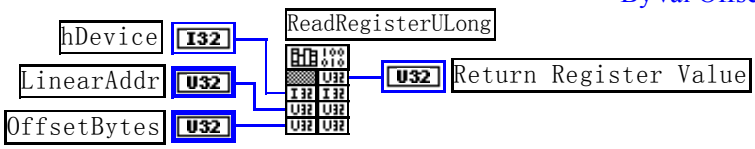
函数原型：

Visual C++:

```
ULONG ReadRegisterULong( HANDLE hDevice,
                         PUCHAR pbLinearAddr,
                         ULONG OffsetBytes)
```

Visual Basic:

```
Declare Function ReadRegisterULong Lib "PXI9100" (ByVal hDevice As Long, _
                                                  ByVal pbLinearAddr As Long, _
                                                  ByVal OffsetBytes As Long) As Long
```



功能：以四字节（即 32 位）方式读 PXI 内存映射寄存器的指定单元。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pbLinearAddr　PXI设备内存映射寄存器的线性基地址，它的值应由GetDeviceAddr确定。

OffsetBytes　相对与LinearAddr线性基地址的偏移字节数，它与LinearAddr两个参数共同确定WriteRegisterULong函数所访问的映射寄存器的内存单元。

返回值：返回从指定内存映射寄存器单元所读取的 32 位数据。

相关函数： CreateDevice　GetDeviceAddr　WriteRegisterByte　WriteRegisterWord　WriteRegisterULong　ReadRegisterByte　ReadRegisterWord　ReadRegisterULong　ReleaseDevice

Visual C++程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr,   PhysAddr, OffsetBytes;
ULONG   Value;
hDevice = CreateDevice(0);   // 创建设备对象
GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0);    // 取得 PXI 设备 0 号映射寄存器的线性基地址
OffsetBytes = 100;    // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
Value = ReadRegisterULong(hDevice, LinearAddr, OffsetBytes);   // 从指定映射寄存器单元读入 32 位数据
ReleaseDevice( hDevice );   // 释放设备对象
```

：

Visual Basic 程序举例:

：

```
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
Dim Value As Long
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, Linearddr, PhysAddr, 0)
OffsetBytes = 100
Value = ReadRegisterULong( hDevice, LinearAddr, OffsetBytes)
ReleaseDevice(hDevice)
```

：

第四节、IO 端口读写函数原型说明

注意：若您想在 WIN2K 系统的 User 模式中直接访问 I/O 端口，那么您可以安装光盘中 ISA\CommUser 目录下的公用驱动，然后调用其中的 WritePortByteEx 或 ReadPortByteEx 等有“Ex”后缀的函数即可。

- **以单字节(8Bit)方式写 I/O 端口**

函数原型：

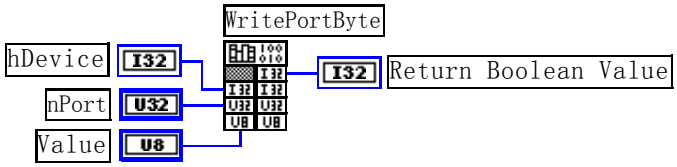
Visual C++:

```
BOOL WritePortByte (HANDLE hDevice,
                    PUCHAR pPort,
                    BYTE Value)
```

Visual Basic:

```
Declare Function WritePortByte Lib "PXI9100" ( ByVal hDevice As Long, _
                                               ByVal pPort As Long, _
                                               ByVal Value As Byte) As Boolean
```

LabVIEW:



功能：以单字节(8Bit)方式写 I/O 端口。

参数：

hDevice 设备对象句柄，它应由CreateDevice创建。

pPort 设备的 I/O 端口号。

Value 写入由 pPort 指定端口的值。

返回值：若成功，返回TRUE，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码。

相关函数： CreateDevice WritePortByte WritePortWord WritePortULong ReadPortByte ReadPortWord

- **以双字(16Bit)方式写 I/O 端口**

函数原型：

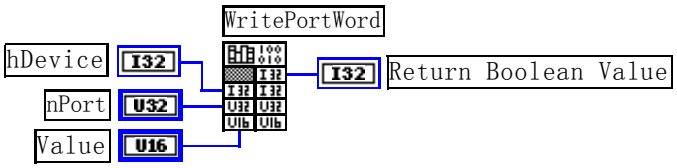
Visual C++:

```
BOOL WritePortWord (HANDLE hDevice,
                    PUCHAR pPort,
                    WORD Value)
```

Visual Basic:

```
Declare Function WritePortWord Lib "PXI9100" (ByVal hDevice As Long, _
                                              ByVal pPort As Long, _
                                              ByVal Value As Integer) As Boolean
```

LabVIEW:



功能：以双字(16Bit)方式写 I/O 端口。
参数：
hDevice设备对象句柄，它应由CreateDevice创建。
pPort　设备的 I/O 端口号。
Value　写入由 pPort 指定端口的值。
返回值：若成功，返回TRUE，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码。
相关函数： CreateDevice　WritePortByte　WritePortWord
WritePortULong　ReadPortByte　ReadPortWord

- **以四字节(32Bit)方式写 I/O 端口**

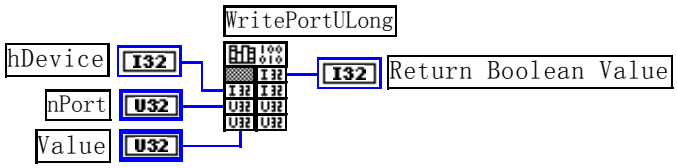
函数原型：
Visual C++:

```
BOOL WritePortULong(HANDLE hDevice,
                    PUCHAR pPort,
                    ULONG Value)
```

Visual Basic:

```
Declare Function WritePortULong Lib "PXI9100" (ByVal hDevice As Long, _
                                               ByVal pPort As Long, _
                                               ByVal Value As Long ) As Boolean
```

LabVIEW:



功能：以四字节(32Bit)方式写 I/O 端口。
参数：
hDevice 设备对象句柄，它应由CreateDevice创建。
pPort　设备的 I/O 端口号。
Value　写入由 pPort 指定端口的值。
返回值：若成功，返回TRUE，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码。
相关函数： CreateDevice　WritePortByte　WritePortWord
WritePortULong　ReadPortByte　ReadPortWord

- **以单字节(8Bit)方式读 I/O 端口**

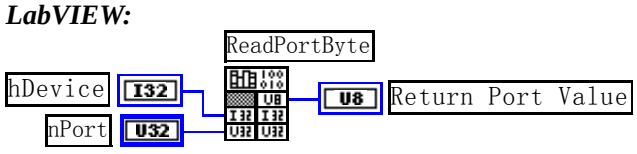
函数原型：
Visual C++:

```
BYTE ReadPortByte( HANDLE hDevice,
                   PUCHAR pPort)
```

Visual Basic:

```
Declare Function ReadPortByte Lib "PXI9100" (ByVal hDevice As Long, _
                                             ByVal pPort As Long ) As Byte
```

LabVIEW:



功能：以单字节(8Bit)方式读 I/O 端口。
参数：
hDevice设备对象句柄，它应由CreateDevice创建。
pPort　设备的 I/O 端口号。
返回值：返回由 pPort 指定的端口的值。
相关函数： CreateDevice　WritePortByte　WritePortWord
WritePortULong　ReadPortByte　ReadPortWord

♦ **以双字节(16Bit)方式读 I/O 端口**

函数原型：

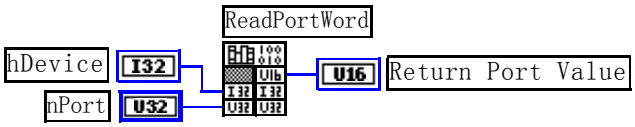
Visual C++:

```
WORD ReadPortWord(HANDLE hDevice,
                  PUCHAR pPort)
```

Visual Basic:

```
Declare Function ReadPortWord Lib "PXI9100" ( ByVal hDevice As Long, _
                                              ByVal pPort As Long ) As Integer
```

LabVIEW:



功能：以双字节(16Bit)方式读 I/O 端口。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pPort　设备的 I/O 端口号。

返回值：返回由 pPort 指定的端口的值。

相关函数： CreateDevice　WritePortByte　WritePortWord　WritePortULong　ReadPortByte　ReadPortWord

♦ **以四字节(32Bit)方式读 I/O 端口**

函数原型：

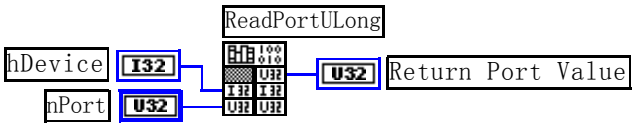
Visual C++:

```
ULONG ReadPortULong(HANDLE hDevice,
                    PUCHAR pPort)
```

Visual Basic:

```
Declare Function ReadPortULong Lib "PXI9100" ( ByVal hDevice As Long, _
                                               ByVal pPort As Long ) As Long
```

LabVIEW:



功能：以四字节(32Bit)方式读 I/O 端口。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

pPort　设备的 I/O 端口号。

返回值：返回由 pPort 指定端口的值。

相关函数： CreateDevice　WritePortByte　WritePortWord　WritePortULong　ReadPortByte　ReadPortWord

第五节、线程操作函数原型说明

♦ **创建内核系统事件**

函数原型：

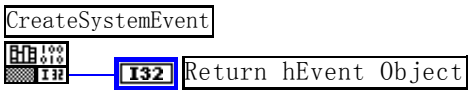
Visual C++:

```
HANDLE CreateSystemEvent(void)
```

Visual Basic:

```
Declare Function CreateSystemEvent Lib " PXI9100 " () As Long
```

LabVIEW:



功能：创建系统内核事件对象，它将被用于中断事件响应或数据采集线程同步事件。

参数：无任何参数。
返回值：若成功，返回系统内核事件对象句柄，否则返回－1(或 INVALID_HANDLE_VALUE)。

◆ **释放内核系统事件**

函数原型：
Visual C++:
BOOL ReleaseSystemEvent(HANDLE hEvent);
Visual Basic:
Declare Function ReleaseSystemEvent Lib " PXI9100 " (ByVal hEvent As Long) As Boolean
LabVIEW:
请参见相关演示程序。

功能：释放系统内核事件对象。
参数：hEvent　被释放的内核事件对象。它应由CreateSystemEvent成功创建的对象。
返回值：若成功，则返回 TRUE。